AD-A037 201 WHARTON SCHOOL OF FINANCE AND COMMERCE PHILADELPHIA P--ETC F/G 9/2
AN ALERTING SYSTEM FOR A DATABASE MANAGEMENT SYSTEM.(U)
DEC 76 R CORTES N00014-75-C-0462
UNCLASSIFIED 77-01-06 NL

| OF |
AD
A037201

END

DATE
FILMED
4-77

(11) B.S.

AN ALERTING SYSTEM FOR A DATABASE MANAGEMENT SYSTEM

Ricardo Cortes

77-01-06

A thesis submitted to the faculty of the Moore School of Electrical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Engineering (for graduate work in Computer and Information Sciences)

University of Pennsylvania
Philadelphia, Pennsylvania

December 1976

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER (14) 77-01-06	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) (6) An Alerting System for a Database Management System.	5. TYPE OF REPORT & PERIOD COVERED (9) final rept.,
6. PERFORMING ORG. REPORT NUMBER 77-01-06	
7. AUTHOR(s) (10) Ricardo Cortes	8. CONTRACT OR GRANT NUMBER(s) (15) ONR Contract N00014-75-C-0462
9. PERFORMING ORGANIZATION NAME AND ADDRESS Decision Sciences Department University of PA/Wharton School Philadelphia, PA 19104	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Technical report
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Information System Arlington, Virginia 22217	12. REPORT DATE 12/76
	13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) (11) Dec 76 (12) 45p.	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Alerting
Database management
DBTG

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes an alerting system for a Database Management System which provides the end users with the facilities to monitor in a dynamic fashion the changes being made on the information, in order to perform some predetermined actions whenever certain conditions become true. The work includes the description of a simplified implementation of an alerting system made on the Wharton Alerting Network Database (WAND) which gives the foundation for a full implementation on a properly shared database system of the kind described by CODASYL DATABASE TASK GROUP in its April 71 report.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

408 757

LB

University of Pennsylvania
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING

AN ALERTING SYSTEM FOR A DATABASE MANAGEMENT SYSTEM

Abstract:

Due to the high volatility of large shared databases, the database users are interested in monitoring certain conditions that may be created as the information is modified by updating programs.

This thesis describes an alerting system for a Database Management System, which provides the end users with the facilities to monitor in a dynamic fashion the changes being made on the information, in order to perform some predetermined actions whenever certain conditions become true. The work includes the description of a simplified implementation of an alerting system made on the Wharton Alerting Network Database (WAND) which gives the foundation for a full implementation on a properly shared database system of the kind described by CODASYL DATABASE TASK GROUP in its april 71 Report (DBTG systems). WAND is an implementation of a DBTG system.

Master of Science in Engineering
(for graduate work in Computer and Information Sciences)

December 1976

Ricardo Cortes
Author

Dr. Howard Lee Morgan
Faculty Supervisor

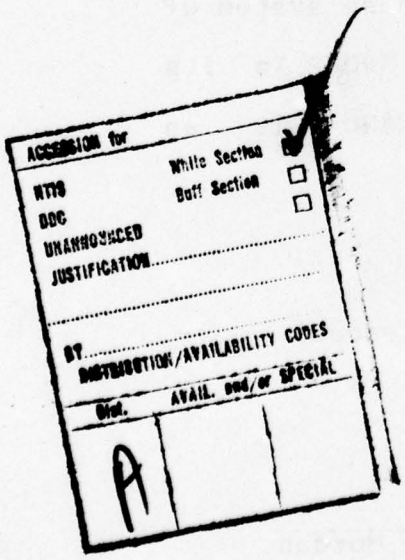
Acknowledgments:

The author wishes to thank Dr. Peter Buneman who suggested the subject of this work and whose advise and supervision has been most helpful.

Special thanks are due to Dr. Noah S. Prywes and Dr. Howard Morgan for their encouragement and invaluable help.

Finally, the help of Dr. Robert Gerritsen, Jim Ribeiro, Ruth Zowader, and David Root in the understanding and use of the WAND system and the DEC-System-10 is fully appreciated.

Ricardo Cortes.



CHAPTER 1.

The concept of alerters should become increasingly important as Database technology continues to grow.

The changes being made by programs in the values of items in the database may cause the creation of certain conditions in which one or more users are interested, so that particular actions associated to such conditions are performed.

Conceptually, an alerter can be described as a program that continuously monitors the contents of the database, in order to take action whenever a previously specified condition becomes true.

For example, in a database containing information about the checking accounts of the clients of a bank, an alerter could be used to monitor the account balances in order to print a message for the manager of a branch when one of the client's balance falls below 200 dollars. In this example, the alerting condition is for the checking account balance to be less than 200 and the associated program prints a message with the data of the corresponding account.

An alerting system for a database should provide the end user with the facilities to specify alerters and place them in the database. By specifying an alerter we mean the definition, in a language for that special purpose, of the conditions to be monitored and the corresponding actions to be taken whenever such conditions become true. It should also provide with the mechanism to monitor changes in the database and to trigger the procedures associated to those conditions as defined by the database alerters.

Since the definition of alerters, including the description of alerting conditions and associated programs has to be stored in the database, we have to extend the definition of such database to accept this additional information.

The wide acceptance and use of database management systems have permitted a fast development and constant improvement of such systems in recent years; however, no database management system provides an alerting system to meet the needs of a growing number of users.

Although some programming languages provide the user with the facility to monitor changes in the information contained in the database (for example, the 'ON' statement in PL1 and the 'USE' condition in ASAP

and some COBOLS), they do not meet user's needs, since the definition of alerters is made in the user's programs and it is not stored in the database.

Howard Morgan [1] introduced the concept of alerters in the context of Management Information Systems. Morgan describes alerters as interrupts used to signal to the supervisor the occurrence of conditions which require a program to be executed; the interrupt generating conditions are Boolean conditions on variables or items in the database.

Some special purpose languages like PLANNER [2] and CONNIVER also have incorporated similar concepts; this type of language permit the manipulation of rich databases, but they are limited to deal with small databases and are not designed to handle hierarchical or network data structures.

Another use of a concept similar to alerter has been implemented as DEMONS in the system LDEMON [3], a system for creating and updating simple databases and monitoring changes in the data. LDEMON, however, is not a production system, since it is written in LISP, an interpretive language, and it was mainly programmed for experimental purposes.

In the present work, we intend to provide a basis for the development of an alerting system for a database management system oriented to the manipulation of large, network structured files; our main concern is the kind of DBMS as described by CODASYL's DATABASE TASK GROUP in its april 1971 report [4].

These systems are called DBTG systems and are intended to be programmed on a host language such as COBOL or FORTRAN. DBTG systems consist of a Data Description Language (DDL), used for describing the database, and a Data Manipulation Language (DML), which acts as an interface between the user's programs and the database.

A DBTG system represents database information as a network in which the nodes are individual record occurrences and the record occurrence is the unit of access. The relationship between record types is called a set type, which has one record type designated as its owner and one or more record types defined as the set members. A network database permits the representation of many-to-many correspondences among records in a direct manner by the use of a confluent hierarchy in which the two related records are the owners of two different set types, of which a third

record type (called the 'base' record) is a member.

The smallest unit of named data in DBTG is called a data-item, and an occurrence of a data-item is a representation of its value.

A partial implementation of a DBTG system was made at the Wharton School [5], to experiment with database alerting, as well as interactive schema management, multiuser sharing, and easy restructuring [6].

The Wharton Alerting Network Database (WAND) is implemented in FORTRAN IV along with a small number of MACRO assembly language subroutines on the DECSYSTEM-10.

In the next chapter we describe a simplified alerting system that was implemented on WAND in order to investigate about some problems involved in full implementations of alerting systems on DBTG.

CHAPTER II.

In order to experiment with the implementation of alerters into a DBTG system, WAND has been provided with a simplified version of an alerting system.

The alerting system described in this chapter permits the operation of simple alerters in which the alerting condition as well as the associated action, are limited; alerters are defined to monitor one or more items within the record types described in the schema, so that a change made to the value of any of such items, will trigger a program which must create record occurrences of a special record type, in order to keep in the database the information contained in the modified record both, before and after the modification has been made.

In this simplified alerting system, the evaluation of conditions is not supported, but it is the responsibility of the user to write programs which have to read the information kept by the system and perform the desired evaluations.

This system has been designed in such a way that it provides the foundation for the implementation of a more complete system as described in the next chapter,

in which the alerting condition can take the form of a logical expression involving the values of current item names as well as constant values.

As it will be seen in the next chapter, the conditions in which we are interested should be evaluated by only analyzing the information in the record which is object of a modification before and after such modification is performed.

The alerting system should permit the dynamic change of the conditions being monitored; therefore, those conditions which are true before they are declared in the database, should not be considered in the evaluation. Similarly, if a record is modified and a given alerting condition was true before the modification is made, the alerter should not be triggered even though the condition is also true after the modification.

For these reasons it is necessary to analyze the old and the new information in the record being modified, so that an alerter is not activated when the condition holds for the old version of the record.

For example, if an alerting condition is described as BALANCE LESS THAN 200, and a program makes a modification to this item in a record that has the value of 180 to the value of 160, then the associated action of printing a message that the balance for this particular record has fallen below 200, was made when this balance became 180 and therefore it should not be performed again.

This implementation provides both versions of the record being monitored by alerters, so that in a further development of the system, they can be the input of a processor which should do the evaluation of the alerting condition.

Similarly, the result of this evaluation might be the input for a second processor which should retrieve and interpret the description of the program associated to the alerting condition, and whose execution should be triggered as the condition becomes true.

Because of the great connection that exists between the way the alerting system is implemented and the way the database is shared, some differences may arise when doing a more complete implementation of the alerting system. However, it is important to note that, although WAND does not support proper database

sharing, the alerting system has been designed to include some of the most important features that should integrate any other implementation of alerting systems for shared databases, with just minor differences as needed.

Three programs integrate the alerting system:

-FDPA (File Definition Processor for an Alerting Database). This program, as WAND's FDP (see [5]), processes the schema definition written in DDL, but it accepts an additional clause in the Schema-Entry, to indicate that the database being defined is going to be used for alerting. The complete WAND schema DDL including the alerting clause is shown in Appendix B.

When the 'ALERTING DATABASE' clause is included, the FDPA adds to the schema the definition of six special records and six special sets, which are used by the alerting system to store the information provided by users in the definition of alerters, as well as that generated by the system itself.

The special data structure is written in Data Description Language in Appendix A, and its use by the system is described below.

-INIALR (Initialize an Alerting Database). It is necessary to run the INIALR program before alerters can be defined into the database; this program reads the schema to retrieve all the record names and item names defined by the user and creates an occurrence of special records REC and ITEM respectively for each of the names found.

We will see later that the declaration of an alerter makes use of the record occurrences created by INIALR, as well as some occurrences of record types USER and ALERTER which are created by the user.

-ALERT. This program is called by the MODIFY routine everytime a change is made in the database by this DML command. ALERT checks which items being modified are monitored by alerters and for each of them it creates two copies of the record, associated to the ALERTER record. The first copy has the same information the record had before the MODIFY routine was called and the second copy is the same version of the record as left in the database by this routine.

The special structure used by the alerting system is represented in a graphical form by the Data Structure diagram in figure 2.1. The boxes in that diagram represent record types which are related with

each other by the named sets which are represented by arrows; a record type pointed by an arrow is the member of the corresponding set type, whereas the box from which the arrow departs shows that the record type is the owner of that set.

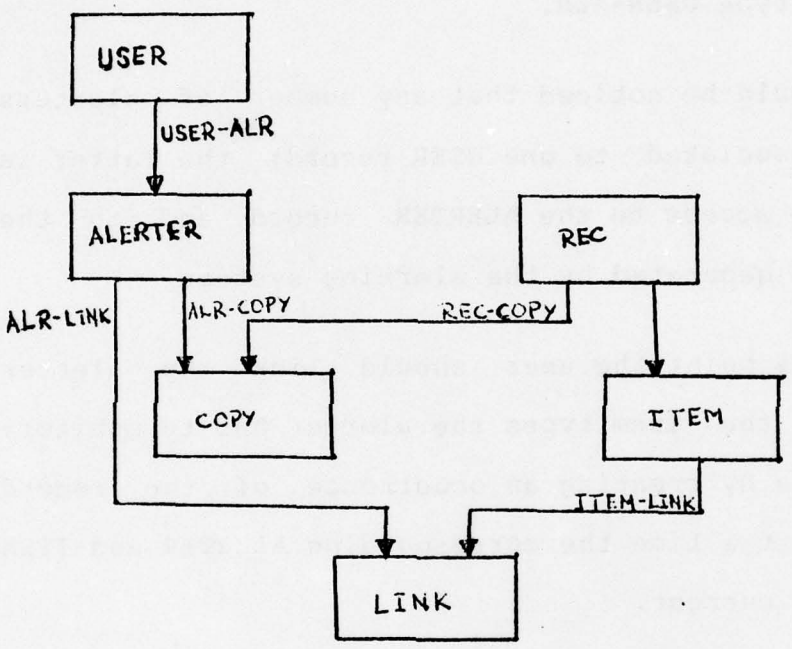


FIGURE 2.1. THE DATA STRUCTURE USED BY THE ALERTING SYSTEM

The record type USER is used to store the password provided by the user to identify his own alerters. Prior to the declaration of alerters by the user, he has to create an occurrence of the USER record type.

The first step in the definition of an alerter consists in creating an occurrence of the special record type ALERTER containing the name of the alerter as the CALC key. The previously created occurrence of the record USER must be current when the ALERTER record is created, so that they become related to each other by the set type USER-ALR.

It should be noticed that any number of alerters can be associated to one USER record; the latter is used as the access to the ALERTER record and to the information generated by the alerting systemn

At this point the user should link the alerter record to the item types the alerter has to monitor; this is done by creating an occurrence of the record type LINK at a time the corresponding ALERTER and ITEM records are current.

One occurrence of LINK has to be created for each item type the alerter is to monitor and it is linked to the alerter record by the set ALR-LINK and to the item record by the set ITEM-LINK. This constitutes an occurrence of a confluent hierarchy in which the record types ALERTER and ITEM keep a many-to-many relationship; LINK is the base record of the confluent hierarchy. In this way we can declare alerters to

monitor one or more item types and each item type can be monitored by different alerters.

An example should be helpful to understand how an alerter is declared in this system; in our example we assume that the record type CLIENT was defined in the schema containing two item types: CHKBAL and SAVBAL.

After running the INIALR program, the database must contain an occurrence of the set type REC-ITEM with two member occurrences named CHKBAL and SAVBAL respectively. The owner of the set is an occurrence of the record REC containing the name of record type CLIENT as shown in figure 2.2. The box marked CLIENT in the diagram is an occurrence of the record type REC and CHCKBAL and SAVBAL are occurrences of the record type ITEM which are stored in the database by the program INIALR; the arrows represent next-pointers of the set type REC-ITEM.

The following routine, written in a hypothetical language introduces the definition of an alerter into the database; the user identification is 'JOHNSON', the name of the alerter is 'LOWBAL' and it is to monitor changes in the item type CHKBAL:

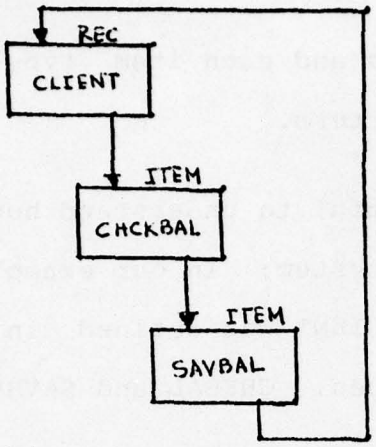


FIGURE 2.2. AN OCCURRENCE OF SET TYPE REC-ITEM

```
BEGIN
     USER IS 'JOHNSON'
     STORE USER
     ALERTER IS LOWBAL
     STORE ALERTER
     ITEM IS CHKBAL
     FIND ITEM
     STORE LINK
END.
```

Notice that the last instruction in the example stores an occurrence of the record type LINK under the appropriate ALERTER and ITEM records, so that two new occurrences of set types ITEM-LINK and ALR-LINK are created as shown in figure 2.3.

Starting from an occurrence of the record type ALERTER and via the associated LINK record occurrences, all the items monitored by an alerter can be retrieved; conversely, all the alerters monitoring a given item can also be obtained by going through the list of LINK records related to the item by the set ITEM-LINK.

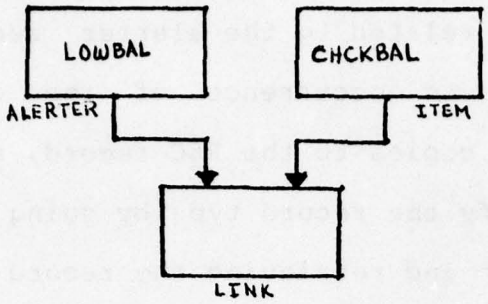


FIGURE 2.3. THE STRUCTURE CREATED BY THE DEFINITION OF AN ALERTER.

The alerting system, by using the information provided in the declaration of alerters, can detect the particular item types that are being monitored by alerters.

The DML MODIFY routine, just before storing back the modified record in the database, calls the program ALERT, which in turn detects which items in the record have been changed in value. For those items, ALERT finds the associated alerters, in order to create a pair of copies of that record linked to the ALERTER record occurrence by the set type ALR-COPY.

The record type COPY is used to store the pairs of copies produced by the triggered alerters; the first copy of the record passed to the MODIFY routine, contains the information in the record before the modification, while the second copy contains the modified information.

Both copies are related to the alerter record by the set ALR-COPY; an occurrence of the set type REC-COPY links these copies to the REC record, so that the user can identify the record type by going back to the owner of this set and retrieving the record name.

An example of the use of alerters in this system is provided in Appendix C, including the description of the programs that evaluate the alerting conditions.

As it can be seen from the description above, this simplified alerting system leaves a great deal of work to the user program; the nature of the alerters handled by the system is also oversimplified. However, it was intended to provide some insight on some problems that can be encountered in implementing a more complete alerting system for DBTG.

In the next chapter we turn our attention to the discussion of more complicated alerters that can be handled by a full implementation of an alerting system; we will also analyze the problem of a shared on line database system, in whose context an alerting system is most useful.

CHAPTER III.

When we refer to an alerting system, we mean a more complete system than that described in the previous chapter, which is a simplification of this concept.

An alerting system for a DBTG has been thought to provide with the facilities for the end user to declare more complicated alerters which should permit him to monitor the creation of predetermined conditions in the database as it is being modified by several programs which share access to the information contained in it.

The concept of alerter surged from the need to dynamically monitor the modifications being made to the information contained in a database from several terminals and at the same time.

For a database in which modifications are made in a centralized manner it might be easy to find cheaper ways to monitor the conditions created in the database as a result of such modifications.

Similarly, when the conditions that have to be monitored, as well as the associated actions are of a fixed nature, it might be more convenient to include them as part of the programs that introduce changes

into the database.

In a properly shared database, an alerting system provides with the facilities to declare into the database the conditions that should be monitored, and such conditions can be constantly modified in accordance to the changing needs of the alerting system users.

Because of the high volatility of many databases as that of an airline reservation system or a Stock Exchange information system, in which hundreds of transactions are made in just few minutes from many different sources, it is possible for the conditions in which one is interested, to change along with the environment to which the system is referred (e.g. to make decisions about rescheduling certain flights or to change the composition of a given portfolio).

Due to the great connection that exists between the alerting system and the database sharing mode, the way a particular implementation of the former is made, very much depends on the way the latter is implemented.

Despite of some minor differences that must be observed from one particular implementation to another, the most important features of an alerting system are

discussed in general terms in the present chapter. Some of these features were successfully implemented for the WAND system and their design can be transferred to other implementations.

The alerting system for a DBTG works as an interface between the users' programs and the database itself, as it accepts the definition of alerters as well as the deletion of previously declared alerters; in performing the alerting mechanism, the alerting system has also interaction with the DML routines which modify the information contained in the database; finally, it makes use of the description of the database contained in the schema, as well as it accesses the information of the database itself.

Figure 3.1 shows the components of the DBTG system and their interaction with the alerting system.

The line marked as PHASE I in the diagram represents the communication between the alerting system and the user's program, which permits the declaration of alerters, as well as deletion and interrogation of previously defined alerters.

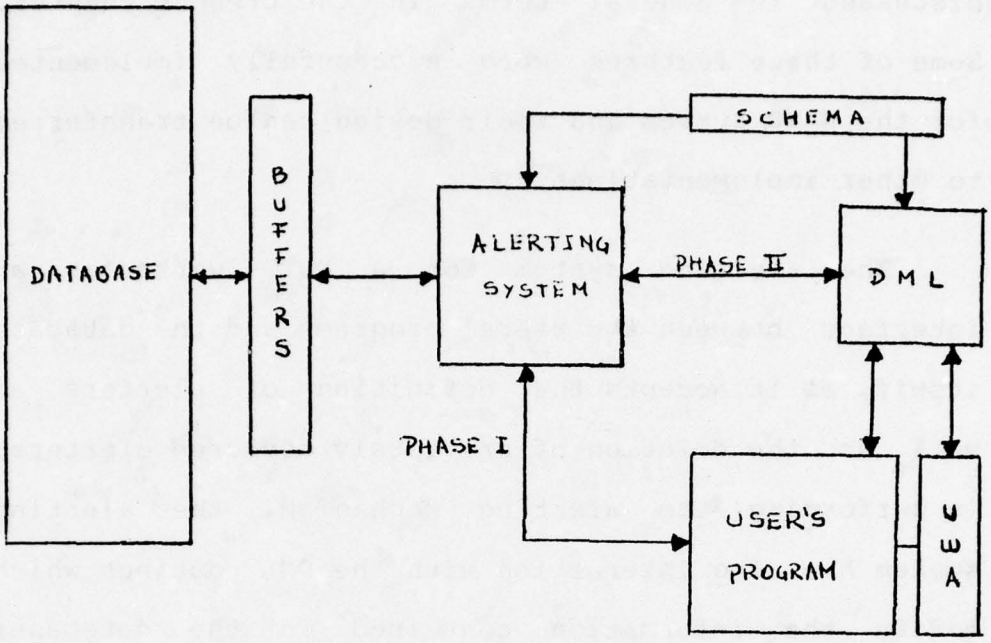


FIGURE 3.1. INTERACTION BETWEEN DBTG AND THE ALERTING SYSTEM.

PHASE II represents the alerting mechanism itself, which constantly monitors the modifications made to the database by DML routines which are in turn called by the user's programs. this second phase includes the evaluation of conditions contained in the alerters description, as well as the triggering of the programs associated to such conditions.

The definition of an alerter is composed of two parts: the alerting condition and the program associated with that condition.

The information provided by the user in the declaration of alerters, as well as that generated by the alerting system as a result of the evaluation of alerting conditions and the triggering of associated programs, has to be stored in the database and be made available to the users' programs.

One of the most important results obtained from the implementation of the alerting system described in the previous chapter is the use of DBTG structures for storing the alerting system information and its manipulation by the DML routines of the DBMS.

As it is suggested by our implementation, the description of the alerting structure into the schema can be automatically made by the program that compiles the definition of the schema, and its creation does not have to be the responsibility of the user.

The special structure designed for our alerting system is general enough to include most of the information that is needed for a full implementation and therefore, the structure needed for other implementations on shared databases should be very similar to the one described in the previous chapter.

The alerting condition, regardless of the way it is represented in the database, should be equivalent to a logical expression with the item types defined in the schema, and constant values as operands. This logical expression can be the input of a processor which, driven by that information, should evaluate the condition represented at the moment the database is modified.

In the alerting system described in the preceding chapter, the DML MODIFY routine calls the program ALERT, which in turn checks in the alerting structure if there are any conditions which become true when the modification is performed; in this case, however, the only condition that the system accepts, is a change in the value of an item being monitored by alerters.

In a more complete alerting system, it should be possible to describe more complicated conditions involving relations among several item types. Although it might seem desirable to be allowed to declare any possible logical condition as an alerting condition, in a real implementation it is necessary to impose some restrictions to the kind of conditions that should be allowed to declare, since not every logical expression involving item types and constant values, can be easily

or even possibly evaluated.

The alerting system is concerned only with those conditions that become true at the moment a record in the database is modified, and not with those which are true before the modification, because of the dynamic nature of alerters, and the fact that the triggering of programs should be done only once; for this reasons, it is necessary to evaluate the alerting condition for the information contained in the record before and after the modification is performed.

We visualize the evaluation of alerting conditions as being made by a processor which can be a program driven by tables, whose input is the description of the condition which has to be evaluated and which is stored in the database. As a result of this, the processor evaluates the condition by operating item values and constant values, to produce a yes-or-no decision about the truth value of the evaluated condition. This process is diagrammed in figure 3.2.

From figure 3.2 we can get an idea of the kind of restrictions that seem necessary for the conditions acceptable by the alerting system.

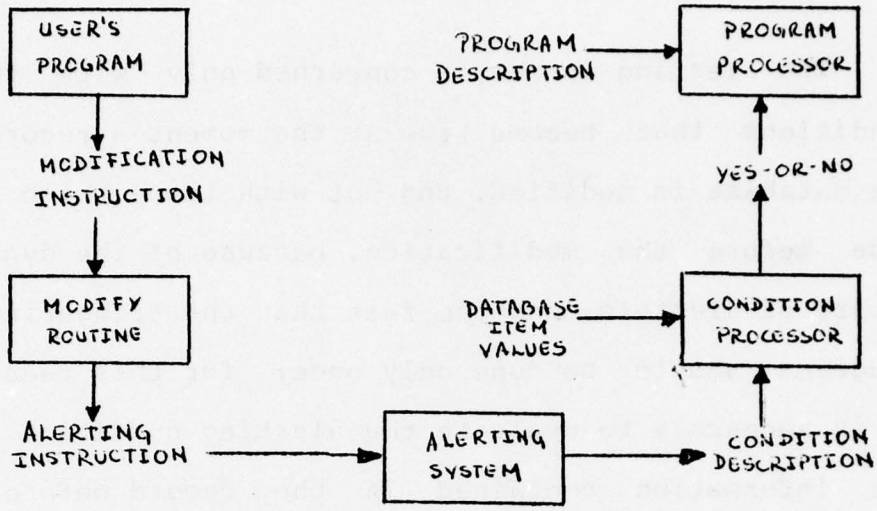


FIGURE 3.2. THE EVALUATION OF AN ALERTING CONDITION.

The evaluation of the alerting condition is directed by the description of such condition which includes item types and constant values, as well as operators; some examples of alerting conditions follow:

1.- (AGE .GT. 21)

2.- (CHECKBAL .LT. 200 .AND. SAVBAL .LT. 200) .OR.
(CHECKBAL + SAVBAL .LT. 400)

3.- (AVGE(INCOME) .GT. 5000)

In these examples we assume that AGE is an item type within a person's record type; CHKBAL and SAVBAL are also item types; and, while INCOME is an item type, AVGE is the name of a function which operates on all the occurrences of a given item type (INCOME in this case), to produce the average of the item values belonging to a given set type.

The three conditions shown above are examples of three different levels of complexity that alerting conditions can be allowed to achieve.

In the first example, AGE is an item type within the record type which is the object of a modification. The evaluation of this condition is performed by only accessing the item occurrence which is available in the system buffers at that moment, and comparing it against the constant value 21.

The condition of the second example includes two item types within the same record type; when a modification is made to this record, the new and old values of items CHECKBAL and SAVBAL are available to the system in core, so that a logical processor can perform the evaluation of this condition.

In the third example, the alerting condition shown involves the value of just one item type, but as compared with the previous examples, its evaluation involves the values of other occurrences of said item type, other than that currently in the system buffers.

In a DBTG system, one occurrence of each record type is current at a given time, hence a condition like that of our third example, requires some additional process to be done, other than that of operating on current item occurrences.

For implementation as well as processing time and space considerations, we find desirable to restrict the class of alerting conditions that can be processed by the alerting system, to those conditions which involve only occurrences of items belonging to a record type, and which are current in the moment the alerting condition has to be evaluated, i.e., when the record is modified.

It is important to note that this restriction will not preclude the capability to perform a process like the one described in the third example, since this can be included in the program associated with that condition.

There exists a trade-off between the cost of the processes needed to evaluate the alerting condition, and that performed by the associated user's program. To illustrate this trade-off, we might consider an alternative to the third condition above, in which any change in the value of the item type INCOME is monitored by an alerter; the associated program should, in our case, go through all occurrences of the record type over which the AVGE function should operate, in order to calculate the average of these values.

This program should in turn trigger the program that in the original case should be called when the alerting condition became true, or it could contain its process.

As suggested by our second example, it should be possible for alerting conditions to include arithmetic as well as Boolean expressions made of items and constant values.

The problems of evaluating the correctness of the logical expression at the moment it is declared into the database, can be solved by some minor modifications to similar processors used by programming languages, in order to provide for some validation of the item types

used in such expressions.

In the alerting system described in the previous chapter, a modification made to an item type which is being monitored by an alerter, results in the creation of two copies of the record type being modified, and containing the information before and after the modification is made. These two copies of the record, in a further development of the alerting system, should be the input for a processor which should evaluate the alerting condition.

The second part of the declaration of an alerter, constitutes the program associated with the alerting condition, which should be triggered when some modification made to the database information causes the alerting condition to become true.

There should be no restrictions for the program that can be associated with an alerting condition; i.e., the associated programs should be allowed to be as complex as required, in order to meet the processing needs of the database users. However, this requires further investigation on the way that programs can be stored in the database and how it can be decided wether a given program should reside in the user's working area or in the database in terms of the size of the

program description.

In most cases, a reduced number of commands including DML statements should be enough to integrate the desired programs. For such cases, it should be possible to store in the database the description of the programs, which should be retrieved and executed by a program processor, as shown in figure 3.2.

Once again, for efficiency considerations, it would be desirable to have small programs stored in the database, and their codification should be made from a small number of selected commands, so that a simple processor can accept this description in order to execute it.

The set of commands referred above could be like that provided by DBLOOK in the WAND system (see [5]). DBLOOK is an interactive processor which provides access to the database by accepting and executing any DML command.

DBLOOK has several additional commands that assist in the data access, like DISPLAY, REPEAT, and assignment commands as well as interrogation about sets, records and items defined in the schema.

The set of commands provided by DBLOOK has proven to be sufficient to perform extensive data manipulation to fulfill the most common process requirments in a database.

In addition to storing programs in the database, the alerting system should also provide with the facilities to read the program associated to an alerting condition from a file outside the database.

This external file may either contain a set of commands in the special language accepted by the alerting system, or it can be a program in the host language as a FORTRAN or COBOL program and which is triggered by the system as the alerting condition becomes true.

The use of the indirect file capability should permit the declaration of as large programs as might be needed, without having to use the storage space which has been reserved for other kind of data.

APPENDIX A.

The special structure used by the alerting system, described in Data Description Language:

```
RECORD NAME IS WWWAUSER
LOCATION MODE IS CALC USING WWWIDUSR
DUPLICATES ARE NOT ALLOWED
WWWIDUSR TYPE IS CHARACTER 10
WWWUSFLG TYPE IS CHARACTER 5.

RECORD NAME IS WWWAREC
LOCATION MODE IS CALC USING WWWIDREC
DUPLICATES ARE NOT ALLOWED
WWWIDREC TYPE IS CHARACTER 10
WWWRCFLG TYPE IS CHARACTER 5.

RECORD NAME IS WWWAALRT
LOCATION MODE IS CALC USING WWWIDALR
DUPLICATES ARE NOT ALLOWED
WWWIDALR TYPE IS CHARACTER 10
WWWALRFG TYPE IS CHARACTER 5.

RECORD NAME IS WWWAITEM
LOCATION MODE IS CALC USING WWWIDITM
DUPLICATES ARE NOT ALLOWED
WWWIDITM TYPE IS CHARACTER 10
WWWITMFG TYPE IS CHARACTER 5.

RECORD NAME IS WWWACOPY
LOCATION MODE IS VIA WWWRECPY
WWWCPY TYPE IS CHARACTER 25.

RECORD NAME IS WWWALINK
LOCATION MODE IS VIA WWWITLNK
WWWLNK TYPE IS FIXED.

SET NAME IS WWWUSALR
MODE IS CHAIN LINKED TO PRIOR
ORDER IS LAST
OWNER IS WWWAUSER
MEMBER IS WWWAALRT
LINKED TO OWNER.

SET NAME IS WWWRCITM
MODE IS CHAIN LINKED TO PRIOR
```

```
ORDER IS LAST
OWNER IS WWWAREC
MEMBER IS WWWAITEM
linked TO OWNER.

SET NAME IS WWWALLNK
MODE IS CHAIN LINKED TO PRIOR
ORDER IS LAST
OWNER IS WWWAALRT
MEMBER IS WWWALINK
LINKED TO OWNER.

SET NAME IS WWWALCPY
MODE IS CHAIN LINKED TO PRIOR
ORDER IS LAST
OWNER IS WWWAALRT
MEMBER IS WWWACOPY.

SET NAME IS WWWRECPY
MODE IS CHAIN LINKED TO PRIOR
ORDER IS LAST
OWNER IS WWWAREC
MEMBER IS WWWACOPY.

SET NAME IS WWWITLNK
MODE IS CHAIN LINKED TO PRIOR
ORDER IS LAST
OWNER IS WWWAITEM
MEMBER IS WWWALINK
LINKED TO OWNER.
```

APPENDIX B.

The schema DDL as accepted by the File Description Processor for an Alerted Database (FDPA).

In this description the following notation is used:

1. words that must be replaced with a user-defined name or value are in lower case.
2. _ (underline) word or character that must appear.
3. () encloses a phrase that may be ommitted.
4. [] encloses lines from which only one may be used.
5. !! encloses phrases that may be repeated.

```
SCHEMA NAME IS schema-name
     (PRIVACY LOCK IS password)
     (DATABASE SIZE IS integer PAGES)
     (PAGE SIZE IS integer WORDS)
     (ALERTED DATABASE).


RECORD NAME IS record-name
     LOCATION MODE IS
          [VIA set-name
   CALC USING item-name-1
```

```
 DIRECT                  ]
    !item-name-2 TYPE IS
[CHARACTER integer
 FIXED
 REAL                 ]!.

SET NAME IS set-name
     MODE IS CHAIN
 (LINKED TO PRIOR)
     ORDER IS
 [FIRST
  LAST
  NEXT
  PRIOR]
     OWNER IS record-name-1
     MEMBER IS record-name-2
 (LINKED TO OWNER).
```

APPENDIX C.

As an illustration of the use of the alerting system implemented for the WAND system, this appendix shows the way in which programs can be written to evaluate alerting conditions by using the information generated by the alerting system.

For this example we assume that the schema contains a record type 'CLIENT' with the item type 'NAME' as the CALC key and item types 'CHKBAL' and 'SAVBAL' representing the balances of checking and savings accounts of a bank clients respectively.

We also assume that three alerters have been declared into the database: alerter 'LOWCHK' monitors changes made to the item type 'CHKBAL'; alerter 'LOWSAV' monitoring changes in item type 'SAVBAL'; and alerter 'LOWSUM' which monitors changes made to either item type 'CHKBAL' OR 'SAVBAL'.

Associated to these alerters, three programs with the same names respectively have been written, and they are described in a hypothetical language below:

```
PROGRAM LOWCHK(OLD-CHKBAL,NEW-CHKBAL)
BEGIN
     if OLD-CHKBAL Greater than 200 and
        NEW-CHKBAL Less than 200 then
         print NAME 'CHECKING ACCOUNT BALANCE TOO LOW'
END PROGRAM LOWCHK.


PROGRAM LOWSAV(OLD-SAVBAL,NEW-SAVBAL)
BEGIN
    if  OLD-SAVBAL Greater than 200 and
        NEW-SAVBAL Less than 200 then
         print NAME, 'SAVINGS ACCOUNT BALANCE TOO LOW'
END PROGRAM LOWSAV.


PROGRAM LOWSUM(OLD-CHKBAL,OLD-SAVBAL,
        NEW-CHKBAL,NEW-SAVBAL)
BEGIN
    if OLD-CHKBAL + OLD-SAVBAL Greater than 500 and
       NEW-CHKBAL + NEW-SAVBAL Less than 500 then
         print NAME, 'ACCOUNTS BALANCE TOO LOW'
END PROGRAM LOWSUM.
```

The three programs described above have as arguments the old and new values of the items that have been modified by other programs.

The alerting system, as a result of the modifications made to item types CHKBAL and SAVBAL creates occurrences of the record type COPY, containing the record CLIENT before the modification was made and after it.

There is a fourth program called CHKALERT, which is in charge to trigger the proper routine, in order that the desired condition be evaluated. CHKALERT constantly analyzes the occurrences of record type COPY under the mentioned ALERT records, calling the corresponding program when a pair of COPY records is found.

```
PROGRAM CHKALERT(ALERTER)
DO FOR EVER
BEGIN
    wait for ALERT-WAKE
    find ALERTER record
    For all COPY records DO
```

```
    BEGIN
        If (ALERTER is LOWCHK) then
            call LOWCHK(OLD-CHKBAL,NEW-CHKBAL)
        If (ALERTER is LOWSAV) then
            call LOWSAV(OLD-SAVBAL,NEW-SAVBAL)
        If (ALERTER is LOWSUM) then
            call LOWSUM(OLD-CHKBAL,OLD-SAVBAL,
                        NEW-CHKBAL,NEW-SAVBAL)
    END
END PROGRAM CHKALERT.
```

The program CHKALERT has as argument the name of the condition that has to be evaluated. Although the programs described above should be somewhat different in the WAND system which uses FORTRAN as host language, they are written in an English-like language in order to make them easy to understand for those who are not familiar with that language.

It is now possible to monitor the occurrence of the three conditions described above, namely: (1) CHKBAK LESS THAN 200, (2) SAVBAL LESS THAN 200, and (3) CHKBAL + SAVBAL LESS THAN 500.

Let's suppose that the program CHKALR is executed in a terminal A and that modifications are being made to the database from another terminal B at the same time. The user in terminal A can choose the conditions he wants to monitor by assigning the corresponding name to the alerting inquiry made by the CHKALR program. This program is going to analyze the contents of the database in a continuous way, triggering each time the evaluation of the corresponding condition (either 'LOWCHK', 'LOWSAV', or 'LOWSUM').

Whenever a modification is made from terminal B that causes the condition being monitored to become true, the corresponding program will send a message to terminal A with the name in the record 'CLIENT' and the new value of the items. This evaluation can be stopped by interrupting the execution of the program CHKALR.

REFERENCES.

[1] Howard Lee Morgan. "An Interrupt Based Organization for Management Information Systems". Communications of the ACM. December 1970.

[2] Carl Hewitt. "PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot". Massachusetts Institute of Technology. Ph. D. Thesis. 1971.

[3] Stanley F. Cohen. "LDEMON LISP Alerter System (As a DAISY Interface)". Working Paper 76-05-07. Dept. of Decision Sciences. University of Pennsylvania.

[4] CODASYL Data Base Task Group, April '71 Report. Association for Computing Machinery.

[5] Robert Gerritsen, Ricardo Cortes, Jim Ribeiro, Ruth Zowader. "Wharton Alerting Network Database User's Guide". Dept. of Decision Sciences. The Wharton School. University of Pennsylvania.

[6] Howard Morgan and Robert Gerritsen.-"Dynamic Restructuring of Data Bases with Generation Data Structures". Dept. of Decision Sciences, working paper 75-12-02, University of Pennsylvania.

DISTRIBUTION LIST

Department of the Navy - Office of Naval Research

Data Base Management Systems Project

Defense Documentation Center (12)
Cameron Station
Alexandria, VA 22314

Office of Naval Research (2)
Information Systems Program
Code 437
Arlington, VA 22217

Office of Naval Research (6)
Arlington, VA 22217

Office of Naval Research
Code 1021P Branch Office, Boston
495 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, Illinois 60605

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106

New York Area Office
715 Broadway - 5th Floor
New York, NY 10003

Naval Research Laboratory (6)
Technical Information Division
Code 2627
Washington, DC 20375

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, DC 20380

Office of Naval Research
Code 455
Arlington, VA 22217

Office of Naval Research
Code 458
Arlington, VA 22217

Naval Electronics Laboratory Center
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research and
Development Center
Computation & Mathematics Dept.
Bethesda, MD 20084

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, DC 20350

Mr. Kim B. Thompson
Technical Director
Information Systems Division
(OP-911G)
Office of Chief of Naval Operations
Washington, DC 20350

Bureau of Library and
Information Science Research
Rutgers - The State University
189 College Avenue
New Brunswick, NJ 08903
Attn: Dr. Henry Voos

Professor Omar Wing
Columbia University
Dept of Electrical Engineering
and Computer Science
New York, NY 10027